4. Hardware-Supported Data Structures

The TMS34010 supports several data structures at the machine level:

- **Fields** are configurable data structures whose length can be defined within the range 1 to 32 bits. Two field sizes can be defined simultaneously. A field can begin and end at arbitrary bit addresses.

- **Bytes** are a special case of field in which the field length is fixed at eight bits and is sign extended. Bytes can begin on any bit boundary within a word.

- **Pixels** are configurable data structures; pixel length can be programmed to be 1, 2, 4, 8, or 16 bits (always a power of two). Pixels are aligned so that they do not cross word boundaries in memory.

- **Two-dimensional pixel arrays** are rectangular groups of pixels that are manipulated using the PIXBLT (pixel block transfer) and FILL (pixel block fill) instructions. A pixel array can be moved from one area of memory to another in a single PixBlt operation. It can be combined with another array of the same size by performing Boolean or arithmetic operations on the corresponding pixels of the two arrays.

The number of bits in a pixel, field, or array is programmable, but byte length is fixed. Two field sizes and one pixel size can be specified simultaneously. The size and starting addresses of the pixel arrays that are manipulated during a PixBlt operation are specified by the values loaded into dedicated hardware registers.

Topics in this section include:

Section **Page**

4.1 Fields 4-2
4.2 Pixels 4-6
4.3 XY Addressing 4-11
4.4 Pixel Arrays 4-14

4.1 Fields

The TMS34010 supports two software-configurable field types, Field 0 and Field 1. A field in memory is defined by two parameters:

- Starting address
- Field size (1 to 32 bits)

A field's starting address is the address of the field's LSB. A field can begin at an arbitrary bit address in memory. When a field is moved from memory to a general-purpose register, the field is right justified within the register; that is, the field's LSB coincides with the register's rightmost bit (bit 0). The register bits to the left of the field are all 1s or all 0s, depending on the values of both the appropriate FE (field extension) bit in the status register, and the sign bit (MSB) of the field. If FE=1, the field is sign extended; if FE=0, the field is zero extended.

Field size can range from 1 to 32 bits. The lengths of fields 0 and 1 are defined by two 5-bit fields in the status register, FS0 and FS1.

Figure 4-1 illustrates a field in memory. In this example, the field straddles the boundary between words *N* and *N+1* in memory. Field extraction and insertion is performed by on-chip hardware:

- To move the field to a general-purpose register, the TMS34010 extracts the field from memory by reading word *N* and word *N+1* in separate cycles.

- To move the field from a general-purpose register, the TMS34010 inserts the field into memory by reading and writing word *N*, and reading and writing word *N+1*.

The memory operations necessary to insert or extract a field are performed automatically by special hardware, and are transparent to software.

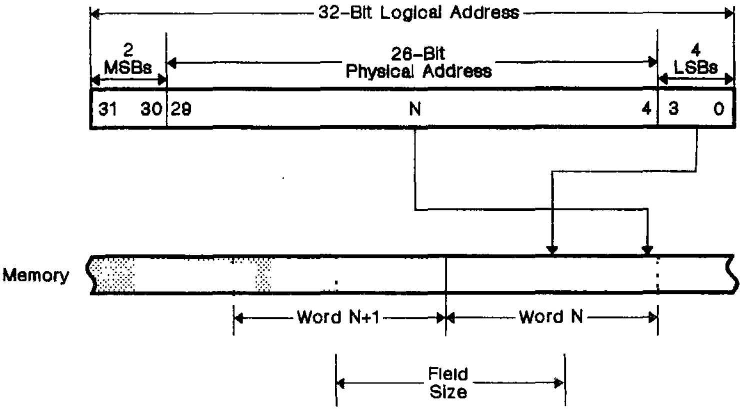


Figure 4-1. Field Storage in External Memory

In Figure 4-1, word *N* is pointed to by a 26-bit physical address output by the GSP to memory. This 26-bit address corresponds to bits 4–29 of the field's 32-bit logical address. The four LSBs of the logical address point to the beginning of the field within word *N*.

The number of memory cycles required to extract or insert a field depends on how the field is aligned in memory. Field manipulation is more rapid when fields are stored in memory so that they do not cross word boundaries. Figure 4-2 illustrates various cases of alignment and nonalignment of fields to word boundaries in memory. Given a field starting address and field length, the memory controller will recognize the specified field alignment as one of the seven cases in Figure 4-2. Field extraction and field insertion are performed in a manner that requires the minimum number of memory cycles.

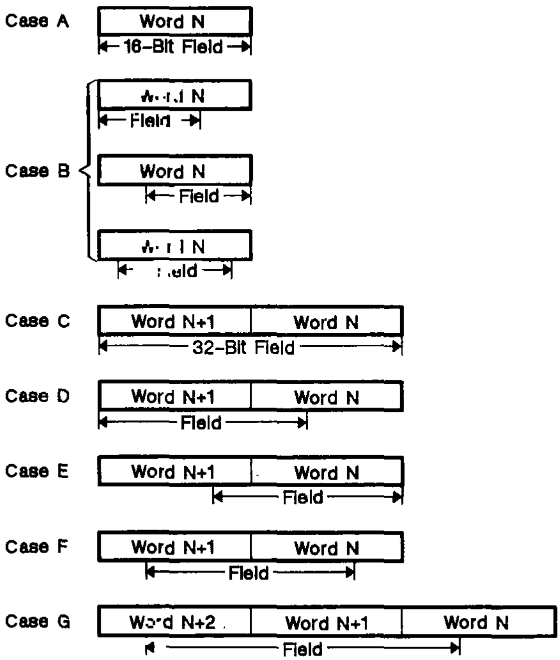


Figure 4-2. Field Alignment in Memory

Case A A 16-bit field is aligned on word boundaries. Field extraction requires a single read cycle, and field insertion requires a single write cycle.

Cases B1–B3 The field length is less than 16 bits.

- In **Case B1**, the field starting address is not aligned to a word boundary, although the end of the field coincides with the end of the word.
- In **Case B2**, the field starting address is aligned to a word boundary, but the end of the field does not coincide with the end of the word.
- In **Case B3**, the field length is 14 bits or less, and neither the start nor the end of the field is aligned to a word boundary.

For Cases B1-B3, a field extraction requires a single read cycle. A field insertion requires the following sequence of memory cycles:

1) Read word *N*
2) Write word *N*

Case C A 32-bit field is aligned on word boundaries. A field extraction requires the following sequence of memory cycles:

1) Read word *N*
2) Read word *N+1*

A field insertion requires the following sequence of memory cycles:

1) Write word *N*
2) Write word *N+1*

Case D The field size is greater than 16 bits. The field starting address is not aligned to a word boundary, but the end of the field coincides with the end of the word. A field extraction requires the following sequence of memory cycles:

1) Read word *N*
2) Read word *N+1*

A field insertion requires the following sequence of memory cycles:

1) Read word *N*
2) Write word *N*
3) Write word *N+1*

Case E The field size is greater than 16 bits. The field starting address is aligned to a word boundary, but the end of the field does not coincide with the end of the word. A field extraction requires the following sequence of memory cycles:

1) Read word *N*
2) Read word *N+1*

A field insertion requires the following sequence of memory cycles:

1) Write word *N*
2) Read word *N+1*
3) Write word *N+1*

Case F The field straddles the boundary between two words. Neither the start nor the end of the field is aligned to a word boundary. A field extraction requires the following sequence of memory cycles:

1) Read word *N*
2) Read word *N+1*

A field insertion requires the following sequence of memory cycles:

1) Read word *N*
2) Write word *N*
3) Read word *N+1*
4) Write word *N+1*

Case G The field size ranges from 18 to 32 bits, and the field straddles two word boundaries. Neither the start nor the end of the field is aligned to a word boundary. A field extraction requires the following sequence of memory cycles:

1) Read word *N*
2) Read word *N+1*
3) Read word *N+2*

A field insertion requires the following sequence of memory cycles:

1) Read word *N*
2) Write word *N*
3) Write word *N+1*
4) Read word *N+2*
5) Write word *N+2*

A field insertion modifies only the portion of a word that lies within a field. The GSP memory controller must perform a read-modify-write operation when a field that does not begin and end on even 16-bit word boundaries is to be written to memory. This occurs when the four LSBs of the address are not 0, or when the specified field size is a value other than 16 or 32. The memory controller uses these two parameters (address LSBs and field size) to produce a mask that identifies the bits in the word corresponding to the field. Hardware uses the mask to perform the read-modify-write cycle. The GSP's local memory control logic automatically generates the the mask and executes the read-modify-write operation; this is transparent to software.

Figure 4-3 shows an example of inserting a 5-bit field stored in a register to logical address >0000 0008.

- In Figure 4-3 *a*, the field to be inserted is shown right-justified in the 16 LSBs of the designated general-purpose register.
- In *b*, memory controller hardware has rotated the field to align it with the destination in memory.
- In *c*, the GSP reads the original word from the destination in memory.
- In *d*, the mask is generated to designate the bits to be modified.
- In *e*, the field is inserted into the word from memory, and the result is written back to the destination address in memory.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(a) Field to be inserted	X	X	X	X	X	X	X	X	X	X	X	F	F	F	F	F
(b) Rotate to align to bit 8	X	X	X	F	F	F	F	F	X	X	X	X	X	X	X	X
(c) Initial destination data	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
(d) Mask generated	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
(e) Final destination data	A	A	A	F	F	F	F	F	A	A	A	A	A	A	A	A

Figure 4-3. Field Insertion

In the more complex case in which a field straddles one or two word boundaries in memory, the portion of the field lying within each word is inserted into that word using the methods described above.

4.2 Pixels

The term pixel has two meanings in the context of a TMS34010-based graphics system. Outside the GSP, a pixel is a picture element on a display surface. Inside the GSP, a logical pixel is a software-configurable data structure supported by the GSP instruction set. The logical pixel data structure in GSP memory contains the information needed to specify the attributes of a picture element visible on a screen. The information for a horizontal line of pixels on the screen is usually stored in consecutive words in memory.

4.2.1 Pixels in Memory

Within GSP memory, the pixel data structure is defined by two parameters:

- Starting address
- Pixel size

A pixel's starting address is the address of the LSB of the pixel.

Pixel size (the number of bits per pixel) is defined in the PSIZE register. A pixel can be 1, 2, 4, 8, or 16 bits long. The GSP treats pixels as a special case of a field in which the field size is constrained to be a power of two. However, pixels do not cross word boundaries within memory; they are aligned within memory so that an integral number of pixels is contained within the boundaries of a memory word. For example, a 2-bit pixel should begin at an even bit address whose LSB is 0, a 4-bit pixel should begin at a bit address whose two LSBs are 0s, and so forth.

When a pixel is moved from memory to a general-purpose register, the pixel is right justified within the register. That is, the LSB of the pixel coincides with the rightmost bit (bit 0) of the register. Register bits to the left of the pixel are loaded with 0s.

Figure 4-4 illustrates pixel storage in memory. The pixel is located within the word pointed to by the 26-bit physical address corresponding to bits 4–29 of the 32-bit logical address of the pixel. The four LSBs of the logical address specify the displacement of the pixel within the word. When the pixel length is less than 16, each word contains two or more pixels.

Pixel extraction and insertion is performed by on-chip hardware in a manner that requires the minimum number of memory cycles. (The operations are transparent to the programmer.) In the worst case, two memory cycles (a read followed by a write) are required to insert a pixel of less than 16 bits. Inserting a 16-bit pixel requires a single write cycle, and extracting a pixel (1 to 16 bits) requires a single read cycle.

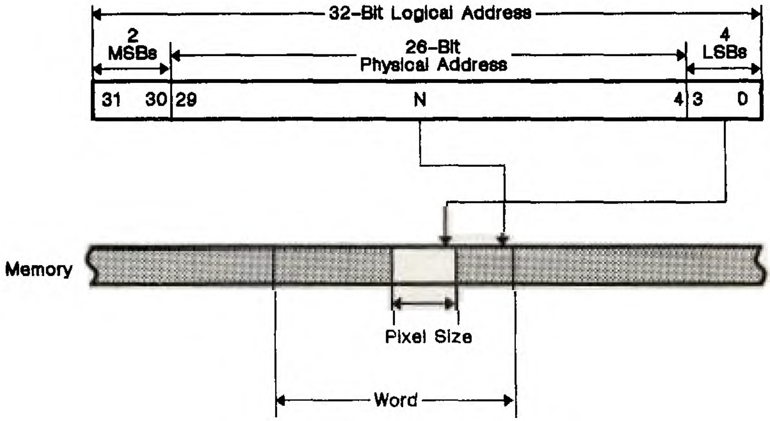


Figure 4-4. Pixel Storage in External Memory

4.2.2 Pixels on the Screen

Figure 4-5 illustrates the mapping of pixels from memory to a display screen. The screen refresh function outputs pixels in the sequence of ascending pixel addresses. However, the electron beam sweeps from the left edge of the screen to the right edge during each horizontal scan interval, so pixels appear on the screen in the opposite order of their representation in memory. That is, the least significant pixel (in terms of bit address) appears on the left, and the most significant pixel appears on the right.

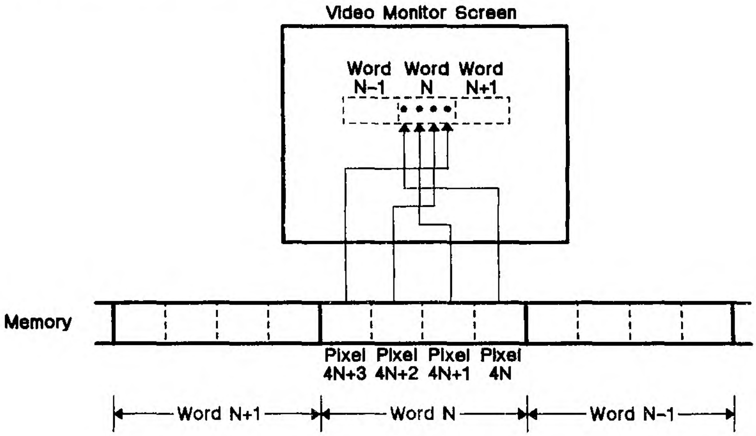


Figure 4-5. Mapping of Pixels to Monitor Screen

The GSP allows a pixel to be identified either in terms of its XY coordinates on the screen, or in terms of the address of the logical pixel in memory. These two methods are called *XY addressing* and *linear addressing*, respectively.

When XY addressing is used, the origin can be selected to lie in either the upper left or lower left corner of the screen. The position of the origin is controlled by the ORG bit in the DPYCTL register. Figure 4-6 *a* illustrates the default coordinate system (ORG=0), in which the origin of the two coordinate axes is located in the upper left corner of the screen. Figure 4-6 *b* shows the alternate coordinate system (ORG=1) in which the origin is located in the lower left corner of the screen.

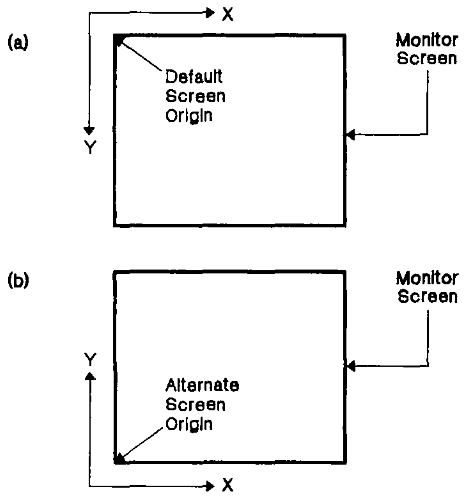


Figure 4-6. **Configurable Screen Origin**

Using the default screen origin, Figure 4-7 illustrates the mapping of pixels from memory to the screen. In Figure 4-7, horizontal movement represents travel in the X direction on the screen. Vertical movement represents travel in the Y direction. The depth of the buffer represents the pixel size. The "on-screen memory" contains the pixels that appear on the screen.

The display memory shown in Figure 4-7 is shown in terms of a "screen format" rather than the "memory format" used in the memory map shown in Figure 3-3 on page 3-4. The screen format places the lowest pixel address at the upper left corner of the memory map. This is the same relative orientation in which pixels appear on the screen. Compare this to the memory format shown in Figure 3-3, which places the lowest bit address at the lower right corner of the memory map. This convention is frequently used in industry to represent the relative location of addresses in memory. In this document, assume the standard memory format is used unless the screen format is indicated.

Figure 4-8 illustrates the mapping of XY coordinates to the on-screen memory. For simplicity, assume that the screen origin coincides with the upper left corner of the display memory. *P* represents the X extent of the display memory and *N* represents the Y extent. Each box represents a pixel within the memory. The number within the box represents the pixel's memory location, relative to the beginning of the on-screen memory. The number in the box is multiplied by the number of bits per pixel to produce the address offset of the pixel from

the start of the display memory. Since the pixel size is constrained to be a power of two, the multiply can be replaced by a simple shift operation.

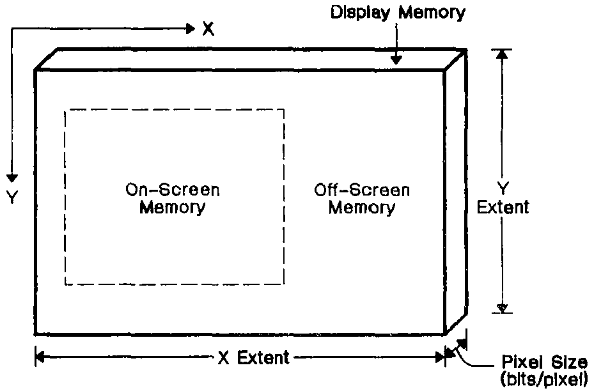
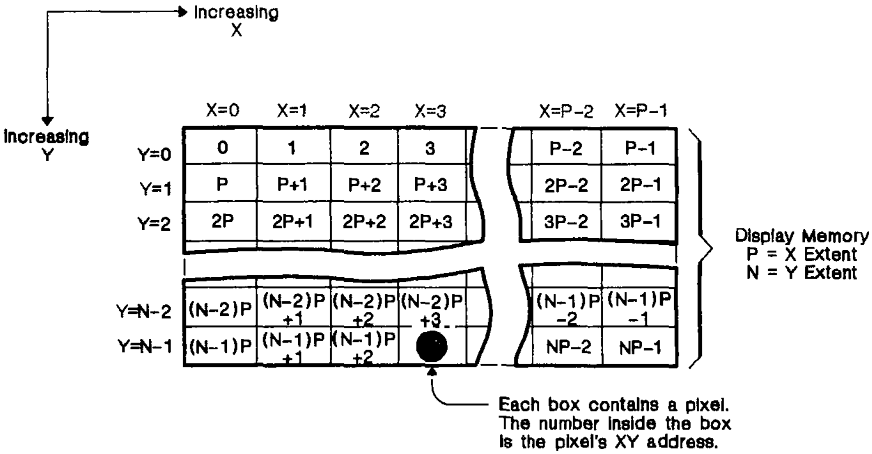


Figure 4-7. Display Memory Dimensions



Display Pitch= (X extent) × (pixel size)
= Differences in 32-bit memory addresses of two vertically adjacent pixels

Figure 4-8. Display Memory Coordinates

4.2.3 Display Pitch

The term *display pitch* refers to the difference in memory addresses between two pixels that appear in vertically adjacent positions (one directly above the other) on the screen. In Figure 4-8, the pitch is calculated as *P* times the pixel size, where *P* is the X extent of the display memory.

The display pitch must be a power of two in order to support XY addressing of pixels on the screen. Linear addressing of pixels on the screen imposes fewer restrictions. In particular, the display pitch for linear addressing may be any value that is a multiple of 16; that is, the four LSBs of the address must be 0s. Of course, features such as automatic window checking are not available with linear addressing.

The pitch of a pixel array is the difference in memory addresses of two vertically adjacent pixels in the array. If the array occupies a rectangular area of the screen, the array pitch is the same as the display pitch.

During a pixel operation such as a PixBlt, the source and destination array pitches are specified in separate dedicated hardware registers. This facilitates the transfer of pixel arrays between on-screen and off-screen memory, which may have different pitches.

A sample display pitch calculation is shown below. In this example, the pixel size is four bits and the X extent of the pixel display is 640 pixels. However, since XY addressing and windowing are to be used, the physical memory is organized so that there are 1024 pixels between successive scan lines. Thus, the X extent of physical display memory is 1024, and the display pitch is:

$$\begin{aligned} \text{Display Pitch} &= (1024 \text{ pixels/line}) \times (4 \text{ bits/pixel}) \\ &= 4096 \text{ (which is } 2^{12}\text{)} \end{aligned}$$

4.3 XY Addressing

The TMS34010 allows pixel addresses to be specified in terms of two-dimensional XY coordinates that correspond to locations on the screen. This is referred to as XY addressing. XY addressing has several benefits:

- TMS34010 software can be easily ported from one display configuration to another. System-dependent details such as the number of bits per pixel and the X extent of the display memory are transparent to the software, but are used by the machine to automatically convert the XY coordinates to the address of a pixel in memory.
- XY addressing allows you to think in terms of the high-level concept of XY coordinates rather than in terms of the machine-level mapping of pixels into memory.
- XY addressing facilitates such functions as window clipping.

Figure 4-9 illustrates XY addressing format. The XY address is stored in a 32-bit general-purpose register. The X and Y components are each treated as 16-bit signed integers. The X component resides in the 16 LSBs of the register, and is right justified to bit 0 of the register. The Y component occupies the 16 MSBs of the register, and is right justified to bit 16 of the register. XY coordinates in the range (-32768,-32768) to (+32767,+32767) can be represented. The clipping window, which identifies the pixels that can be altered during drawing operations, is restricted to positive X and Y coordinate values, (0,0) to (+32767,+32767). Thus, pixels identified by negative X or Y coordinates must always lie outside the window.

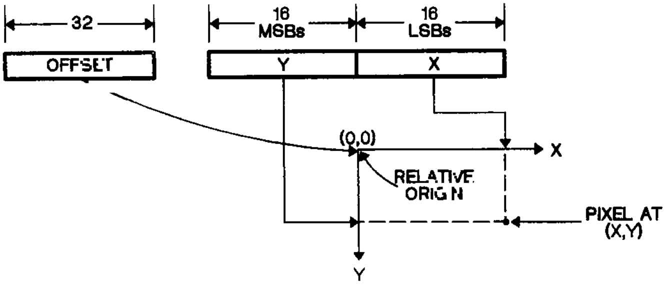


Figure 4-9. Pixel Addressing in Terms of XY Coordinates

4.3.1 XY-to-Linear Conversion

The TMS34010 automatically converts a pixel's XY address to a 32-bit logical address (linear address) for all instructions that use XY addressing. Three parameters are used to perform XY-to-linear conversion:

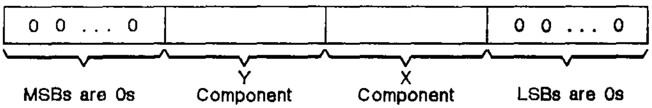
- The logical pixel size (stored in the PSIZE register)
- A pitch conversion factor (stored in the CONVSP or CONVDP registers)
- An offset defining the XY origin (stored in the OFFSET register)

The GSP uses the following formula to calculate the physical address associated with the XY address:

Address = [(Y × display pitch) OR (X × pixel size)] + offset

Since the display pitch and pixel size are both powers of two, the calculation is performed using only shift, OR, and add operations. Window clipping may be used to detect out-of-bounds (negative) X or Y values before this calculation is performed.

Linear addresses are formed from XY addresses by simply concatenating the binary numbers that represent the X and Y coordinate values, as shown in Figure 4-10. The number of 0s to the right of the X component of the address depends on the number of bits per pixel, and equals $\log_2$(pixel size). The displacement of the Y component within the 32-bit logical address in Figure 4-10 is equal to $\log_2$(display pitch). Finally, a 32-bit offset is added to the address in Figure 4-10 to calculate the address in memory of the pixel at coordinates (X,Y). The offset corresponds to the linear address in memory of the pixel at (0,0).



Note: The shift value for the Y component is contained in CONVSP or CONVDP register, depending on the instruction being executed.

Figure 4-10. Concatenation of XY Coordinates in Address

The GSP uses the pitch conversion factors *CONVSP* and *CONVDP* to compute the displacement of the Y component within the address, as shown in Figure 4-10. The Y component is displaced from bit 0 of the address by an amount equal to $\log_2$(pitch), which the hardware obtains by inverting the five LSBs of the appropriate CONVSP or CONVDP register. These values must be loaded through software before executing an instruction that uses XY addressing. CONVSP (source address pitch) is used if the XY address points to a *source* pixel or pixel array; CONVDP (destination address pitch) is used if the XY address points to a *destination* pixel or pixel array. The pixel size stored in the PSIZE register is used similarly to determine the displacement of the X component, as shown in Figure 4-10.

The OFFSET register contains the linear memory address of the pixel located at coordinates (0,0) on the monitor screen. The OFFSET register is used in translating XY coordinates into linear addresses, but does not control which region of the display memory is output to refresh the video screen. It is a virtual screen origin. It allows the coordinate axes of the XY address to be translated to an arbitrary position in memory. The OFFSET register supports the use of "window relative" addressing in which the X and Y coordinates are specified relative to coordinate offsets in the display memory. The position and size of a window can be specified arbitrarily. A new offset specified in terms of XY coordinates can be converted to a linear address using the CVXYL instruction. CVXYL converts an XY address to a linear address for the purpose of absolute memory addressing, or to use special features available to instructions that use linear addressing. Figure 4-11 illustrates the XY-to-linear conversion process.

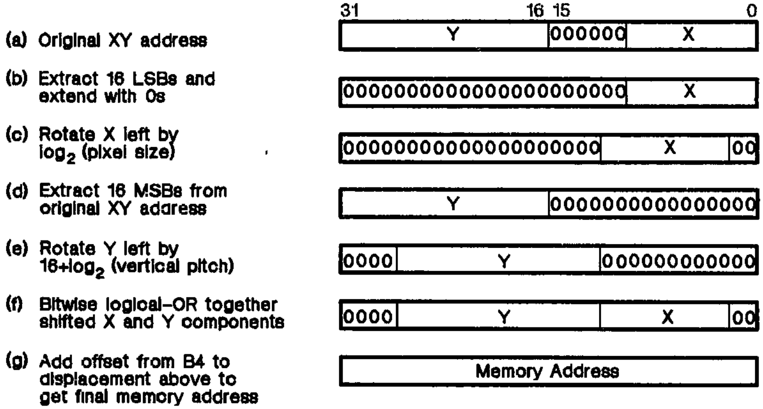


Figure 4-11. Conversion from XY Coordinates to Memory Address

- Step *a* shows the original XY address.
- The X component is extracted in step *b*.
- In step *c*, the X component is shifted left by $\log_2$(pixel size). The result of step *c* represents the product of the X component and the pixel size.
- The Y component is extracted in step *d*.
- In step *e*, the Y component is rotated left by 16+$\log_2$(display pitch). The result of step *e* is Y multiplied by the display pitch.
- In step *f*, the results of steps *c* and *e* are bitwise-ORed to form the displacement in memory of the pixel at (X,Y) from the pixel at the origin.
- In step *g*, the offset is added to produce the final memory address.

The example of Figure 4-11 corresponds to a pixel size of four bits and a pitch of 4,096. The six MSBs of the X half of the XY address (bits 10–15) in Figure 4-11 must be 0s to produce a valid memory address. For this example, the clipping window should be set to disable writes to pixels having X coordinate values outside the range 0 to +1023.

Generally, given a display with a pitch of 2^n, a valid memory address is produced by the XY translation process shown in Figure 4-11 when only the *n* LSBs of the X half of the XY address are nonzero (that is, when the 15-*n* MSBs are 0). X values may be in the range -32768 to +32767 before clipping. However, after clipping, the X value should be a positive number in the range 0 to (X extent -1), where X extent = pitch/pixel size. The GSP's automatic window clipping can be configured to clip pixels lying outside the window; hence, no software overhead is incurred in clipping. Y values lying outside the window are clipped in a similar fashion.

4.4 Pixel Arrays

A rectangular area of the screen that is *DX* pixels wide and *DY* pixels high is an example of a data structure called a *two-dimensional pixel array.* The array contains DX × DY pixels, but can be manipulated by the TMS34010 as one structure. The TMS34010's instruction set includes a powerful set of raster operations, called PixBlts, that manipulate pixel arrays on the screen and elsewhere in memory.

Figure 4-12 shows a pixel array occupying a rectangular region in display memory. The DX pixels in each row of the array are packed together into adjacent cells in the display memory. Rows do not generally occupy adjacent areas of memory, but are separated from each other by a constant displacement called the array pitch. The array pitch is the difference in memory addresses between the start of one row and the start of the row directly beneath it. In the Figure 4-12 example, the array pitch is equal to the display pitch. The product of the array width DX and the pixel size must be less than or equal to the pitch.

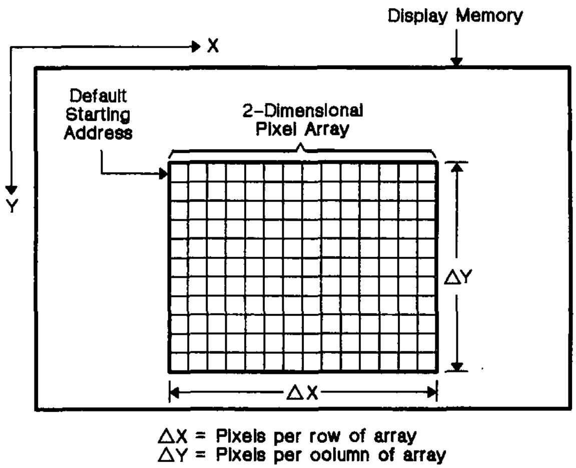


Figure 4-12. Pixel Array

A pixel array is specified in terms of its width, height, pitch, and starting address. The starting address is the address of the first pixel to be moved during a PixBlt. The default starting address is simply the base address of the array; that is, the address of the pixel that has the lowest address in the array.

If as shown in Figure 4-12, the XY origin is located in its default position at the upper left corner of the screen. The default starting address is the address of the pixel located in the upper left corner of the array. When a PixBlt operation moves the pixels from a source pixel array to a destination array, the pixels in each row are moved in sequence from left to right, and the rows are moved in sequence from top to bottom.

Certain PixBlt operations allow the starting pixel to be specified as one of the pixels in the other three corners of the array. This feature is provided so that when the source and destination arrays overlap, the appropriate starting corner can be selected to ensure that no data is lost by being overwritten during PixBlt execution. The order in which pixels in the array are moved can be altered to be from right to left or from bottom to top as appropriate to accommodate the change in starting corner.

The starting address of a pixel array can be specified either in terms of the XY coordinates of the starting pixel (XY address), or the memory address of the starting pixel (linear address):

- An array whose starting location is specified as an XY address is referred to as an *XY array*. In this format, the starting location of the array is identified by the XY coordinates of the first pixel in the array.

- A pixel array whose starting location is specified as a memory address is referred to as a *linear array*. In this format, the location of the array is identified by the memory address of the first pixel (the pixel that has the lowest bit address) in the array.

The XY array format has two advantages. First, the starting location of the array is specified in system-independent Cartesian coordinates rather than as a system-dependent memory address. Second, the GSP's window checking (which allows it to automatically detect an attempt to write a pixel inside or outside a specified window) can only be used in conjunction with XY addressing.

The linear format's main advantage is that the array pitch does not have to be a power of two. This supports a wider variety of memory organizations. Using XY format, the array pitch is constrained to be a power of two.

The general rules governing array pitch are as follows. When an array is specified in XY format, the pitch **must** be a power of two. The pitch for an array specified in linear format may be any multiple of 16; that is, the four LSBs of the pitch must be 0s. There are a few important exceptions to the second rule which are discussed below.

For the special case of a PIXBLT B,XY or PIXBLT B,L instruction, the source pitch may be any value. This feature supports efficient use of memory by allowing adjacent rows of the source array to be packed together with no intervening gaps. The destination pitch must still be a multiple of 16.

Under certain conditions the linear source array specified for a PIXBLT L,XY or PIXBLT B,XY must have a pitch that is a power of two. This is necessary when the linear start address for the array has to be adjusted in the Y direction due to one of the following conditions:

- The source array is automatically preclipped to lie within a rectangular window.

- One of the lower two corners of the source array (refer to Figure 4-12) is selected to be the start address.

In either case, the start addresses specified for both the source and destination arrays are automatically adjusted, and for this purpose the conversion factors specified in the CONVSP and CONVDP registers must be valid.

While PixBlts are useful for moving arrays from one area of the screen to another, they can also be used to move arrays to the screen from other parts of memory, and vice versa. The pitch for the off-screen pixel array can be specified independently of the pitch for the on-screen array. This permits off-screen data to make efficient use of storage, regardless of the display pitch. On-screen objects may be defined as XY arrays but may be more efficiently stored as linear arrays in off-screen memory. The PIXBLT instructions support the transfer of a linear array to an XY array, and vice versa. PIXBLT instructions can also be used to rapidly move blocks of non-pixel data (ASCII characters, for example) from one location in memory to another.